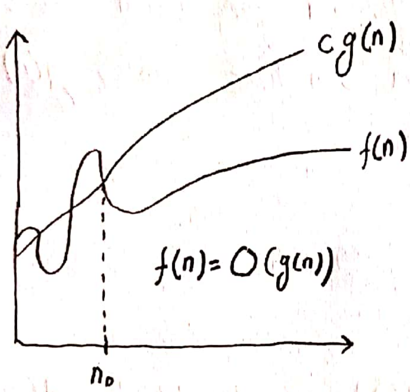
Algorithm Analysis

Asymptotic Notation & Functions & Running Times

We use asymptotic notation primarily to describe the running times of algorithms. Asymptotic notation actually applies to functions, however. The functions to which we apply asymptotic notation will usually characterize the running times of algorithms. Even when we use asymptotic notation to apply to the running time of an algorithm, we need to understand which running time we mean.

O-Notation



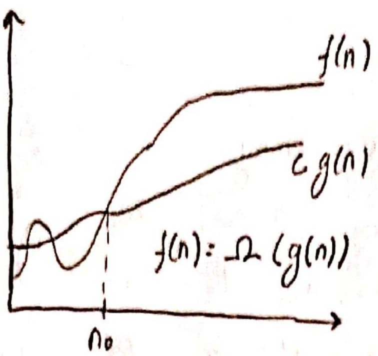
When we have only an asymptotic upper bound, we use O-notation. For a given function $g(n)$, we denote by $O(g(n))$ the set of functions.

$$O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \}$$

Since O-notation describes an upper bound, when we use it to bound the worst case running time of an algorithm, we have a bound on the running time of the algorithm on every input.

- if $f(n)$ is a polynomial of degree d then $f(n)$ is $O(n^d)$
- Use the smallest possible class of functions
  say "$2n$ is $O(n)$" instead of "$2n$ is $O(n^2)$"
- if $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$ then
  - → $d(n) + e(n)$ is $O(f(n) + g(n))$
  - → $d(n) \cdot e(n)$ is $O(f(n) \cdot g(n))$
- if $d(n)$ is $O(g(n))$ and $g(n)$ is $O(f(n))$ then $d(n)$ is $O(g(n))$
- if $p(n)$ is a polynomial in n then $\log p(n)$ is $O(\log n)$

Ω - Notation

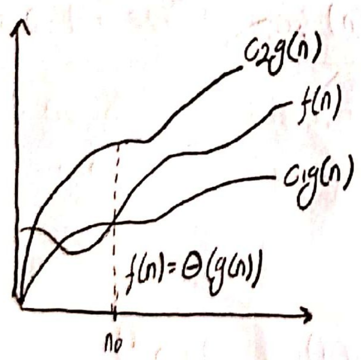


Ω - notation (omega) provides an asymptotic lower bound. For a given function $g(n)$, we denote by $\Omega(g(n))$ the set of functions:

$$\Omega(g(n)) = \{ f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \}$$

When we say that the running time of an algorithm is $\Omega(g(n))$, we mean that no matter what particular input of size n is choosen for each value of n, the running time on that input is at least a constant times $g(n)$, for sufficiently large n.

Θ - Notation



For a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions.

$$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}$$

We say that $g(n)$ is an asymptotically tight bound for $f(n)$.

Examples

- $2n+10$ is $O(n)$

$\rightarrow 2n+10 \leq cn$

$(c-2)n \geq 10$

$n \geq 10/c-2$

pick $c=3$ and $n_0 = 10$

- n^2 is not $O(n)$

$\rightarrow n^2 \leq cn$

$n \leq c$

This inequality cannot be satisfied since c is a constant.

- $7n-2$ is $O(n)$

$\rightarrow 7n-2 \leq cn$

$(7-c)n \leq 2$

pick $c=7$ and $n_0=1$

- $3n^3+20n^2+5$ is $O(n^3)$

$\rightarrow 3n^3+20n^2+5 \leq cn^3$

$20n^2+5 \leq (c-3)n^3$

pick $c=4$, $n_0=21$

- $5n^2$ is $\Omega(n^2)$

$\rightarrow cn^2 \leq 5n^2$

$c \leq 5$

pick $c=5$, $n_0=1$

- $5n^2$ is $\Omega(n)$

$\rightarrow cn \leq 5n^2$

$c \leq 5n$

pick $c=1$, $n_0=1$

- $5n^2$ is $\Theta(n^2)$

$\rightarrow c_1n^2 \leq 5n^2 \leq c_2n^2$

$c_1, c_2 = 5$ and $n_0=1$

- $4n^2+2n$ is $O(n)$

$\rightarrow 4n^2+2n \leq cn$

$4n+2 \leq c$

$n \leq (c-2)/4$

Since c is constant inequality cannot be achieved.

- $3n^2-100n+6 = O(n^2) \rightarrow c=3 \rightarrow 3n^2 > 3n^2-100n+6$
- $3n^2-100n+6 = O(n^3) \rightarrow c=1 \rightarrow n^3 > 3n^2-100n+6 \rightarrow n_0=4$
- $3n^2-100n+6 \neq O(n) \rightarrow cn < 3n^2$ when $n>c$
- $3n^2-100n+6 = \Omega(n^2) \rightarrow c=2 \rightarrow 2n^2 < 3n^2-100n+6 \rightarrow n_0=101$
- $3n^2-100n+6 \neq \Omega(n^3) \rightarrow c=1 \rightarrow n^3 > 3n^2-100n+6 \rightarrow n>3$
- $3n^3 100n+6 = \Omega(n) \rightarrow cn < 3n^2-100n+6 \rightarrow n>100c$

- $3n^2-100n+6 = \Theta(n^2)$, because both O and Ω apply.
- $3n^2-100n+6 \neq \Theta(n^3)$, because only O applies.
- $3n^2-100n+6 \neq \Theta(n)$, because only Ω applies.

- $f(n) = 2n+10$, $g(n) = n$ $\quad$ $f(n) \in O(g(n))$, $g(n) \in O(f(n))$

$\rightarrow 2n+10 \leq cn$
$10 \leq (c-2)n$
$10/(c-2) \leq n$
$c=3,\ n_0=10$

$\rightarrow n \leq c\cdot(2n+10)$
$n \leq 2nc+10c$
$(1-2c)n \leq 10c$
$c=1,\ n_0=0$

- $2n+5$ is $\Theta(n)$

$\rightarrow 2n+5 \geq c_1 n$
$5 \geq (c_1-2)n$
$c_1=2,\ n_0=1$

$\rightarrow 2n+5 \leq c_2 n$
$5 \leq (c_2-2)n$
$c_2=7,\ n_0=1$

- 2^{n+1} is $\Theta(2^n)$

$\rightarrow 2^{n+1} \leq c_1 2^n$
$2\cdot 2^n \leq c_1 2^n$
$c_1=2,\ n_0=1$

$\rightarrow c_2 2^n \leq 2^{n+1}$
$c_2 2^n \leq 2\cdot 2^n$
$c_2=2,\ n_0=1$

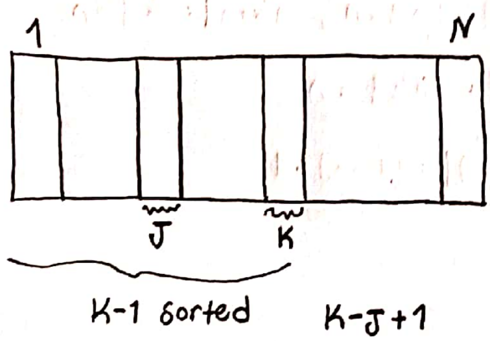
- $(x+y)^2$ is $O(x^2+y^2)$

$\rightarrow (x+y)^2 \leq n(x^2+y^2)$
$x^2+2xy+y^2 \leq nx^2+ny^2$ if no $2xy$ then inequality holds $c \geq 1$
if $x \leq y$ then $2xy \leq 2y^2 \leq 2(x^2+y^2)$
if $y \leq x$ then $2xy \leq 2x^2 \leq 2(x^2+y^2)$
Thus $c=2+1=3$
$(x+y)^2 \leq 3(x^2+y^2)$

Insertion Sort Average Case

worst case $\longrightarrow O(n^2), \Omega(n^2) \longrightarrow \Theta(n^2)$

best case $\longrightarrow O(n), \Omega(n) \longrightarrow \Theta(n)$



Number of operations in inner while

$$\frac{1}{K} \cdot \sum_{J=1}^{K} (K-J+1)$$

$$= \frac{1}{K} \cdot \left(K^2 - \frac{K \cdot (K+1)}{2} + K\right)$$

$$= \frac{2K^2 - K^2 - K + 2K}{2K} = \frac{K+1}{2}$$

$$\sum_{K=2}^{N} \frac{K+1}{2} = \frac{1}{2} \cdot \frac{(N+1) \cdot (N+2)}{2} \longrightarrow \text{Outer loop}$$

- Average Case $= \frac{1}{4}\left[N^2 + 3N + 2\right] \longrightarrow O(n^2)$

Using Limit For Comparing Order Of Growths

$$\lim_{n \to \infty} \frac{t(n)}{g(n)} = \begin{cases} 0, & \text{implies that } t(n) \text{ has a smaller order of growth than } g(n) \\ C \in \mathbb{R}, & \text{implies that } t(n) \text{ has same order of growth as } g(n) \\ \infty, & \text{implies that } t(n) \text{ has a large order of growth than } g(n). \end{cases}$$

- $t(n) = \frac{1}{2} n(n-1)$, $g(n) = n^2$

$$\lim_{n \to \infty} \frac{\frac{1}{2} n \cdot (n-1)}{n^2} = \frac{1}{2} \lim_{n \to \infty} \frac{n^2 - n}{n^2}$$

$$= \frac{1}{2} \lim_{n \to \infty} \left(1 - \frac{1}{n}\right) = \frac{1}{2}$$

Useful Formulus For The Analysis Of Algorithms

Logarithms

- $\log_x y = y^{\log x}$
- $\log xy = \log x + \log y$
- $\log x/y = \log x - \log y$
- $\log_a x = \log_a b \, \log_b x$
- $a^{\log_b x} = x^{\log_b a}$

Floor And Ceiling

- $x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$
- $\lfloor x+n \rfloor = \lfloor x \rfloor + n$, $\lceil x+n \rceil = \lceil x \rceil + n$
- $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$
- $\lceil \log(n+1) \rceil = \lfloor \log n \rfloor + 1$

Summations

* $\sum_{i=L}^{U} 1 = U - L + 1$
* $\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$
* $\sum_{i=1}^{n} i^K \approx \frac{1}{K+1} \cdot n^{K+1}$
* $\sum_{i=1}^{n} 1 = n$
* $\sum_{i=1}^{n} i^2 = \frac{n \cdot (n+1) \cdot (2n+1)}{6}$
* $\sum_{i=1}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}$
* $\sum_{i=1}^{n} i 2^i = (n-1) 2^{n+1} + 2$
* $\sum_{i=1}^{n} \frac{1}{i} \approx \ln n + \gamma$, $\gamma \approx 0.5772$
* $\sum_{i=L}^{U} c a_i = c \cdot \sum_{i=L}^{U} a_i$
* $\sum_{i=L}^{U} a_i \pm b_i = \sum_{i=L}^{U} a_i \pm \sum_{i=L}^{U} b_i$

Mathematical Analysis Of Non-Recursive Algorithms

```
MaxElement(A[0..n-1])
    max ← A[0]
    for i ← 1 to n-1 do
        if A[i] > max
            max ← A[i]
    return max
```

→ Comparison: $A[i] < max$ ①

→ Assignment: $max \leftarrow A[i]$ ②

$$C(n) = \sum_{i=1}^{n-1} 1 = n-1 \in O(n)$$, where n is number of elements.

↓

$C_{worst}(n)$ or $C_{avg}(n)$ or $C_{best}(n)$

```
UniqueElements(A[0..n-1])
    for i ← 0 to n-2 do
        for j ← i+1 to n-1 do
            if A[i] = A[j]
                return false
    return true
```

$$C_{worst}(n) = \sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=1}^{n-2} \left(\sum_{j=i+1}^{n-1} 1 \right)$$

$$= \sum_{i=1}^{n-1} n-i-1 = \sum_{i=1}^{n-1} (n-1) - \sum_{i=1}^{n-1} i$$

$$= (n-1) \cdot \sum_{i=1}^{n-1} 1 - \frac{(n-2)(n-1)}{2} = (n-1)(n-1) - \frac{(n-2)(n-1)}{2}$$

$$= \frac{n \cdot (n-1)}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2)$$

$$T(n) \approx c \cdot C(n) = c + n^2$$

c → cost of basic operations

$T(n)$ → Running time of algorithm

Mathematical Analysis Of Recursive Algorithms

```
factorial (n)
    if n=0
        return 1
    else
        return n * factorial(n-1)
```

$F(n) = F(n-1) \cdot n$

$M(n) = M(n-1) + 1$

$M(n-1)$: to compute $F(n-1)$

$+1$: to multiply $F(n-1)$ by n

Recurrence Relation → Initial Condition: if n=0 return 1 → $M(0) = 0$

no multiplication

The calls stop when $n=0$

$M(n) = M(n-1) + 1$ for $n > 0$

$M(0) = 0$

Method Of Backward Substitution:

$M(n) = M(n-1) + 1$, substitute $M(n-1) = M(n-2) + 1$

$= [M(n-2) + 1] + 1 = M(n-2) + 2$, substitute $M(n-2) = M(n-3) + 1$

$= [M(n-3) + 1] + 2 = M(n-3) + 3$,

⋮

$= M(n) = M(n-i) + i$

→ $i \leftarrow n$

$M(n-n) + n$

$= M(0) + n$

$= n$

```
binaryDigit(n)
    if n = 1
        return 1
    else
        return binaryDigit(⌊n/2⌋)
```

→ Addition: $A(n) = A(\lfloor n/2 \rfloor) + 1$ for $n > 1$

→ Initial Condition: $A(1) = 0$

Smoothness rule ⟶ $n = 2^k$

$$A(2^k) = A(2^{k-1}) + 1 \quad \text{for } n > 1$$

$$A(2^0) = A(1) = 0$$

↳ Backward:

$$A(2^k) = A(2^{k-1}) + 1$$

$$= A(2^{k-2}) + 2$$

$$= A(2^{k-3}) + 3$$

$$\vdots$$

$$= A(2^{k-i}) + i$$

$\hookrightarrow i \leftarrow k$

$$A(2^0) + k = 0 + k = \log_2 n \in \Theta(\log n)$$

- $X(n) = X(n-1) + n$, $X(0) = 0$

$$= [X(n-2) + n-1] + n = X(n-2) + (n-1) + n$$

$$= [X(n-3) + n-2] + (n-1) + n = X(n-3) + (n-2) + (n-1) + n$$

$$\vdots$$

$$= X(n-i) + (n-i+1) + (n-i+2) + \dots + n$$

$i \leftarrow n$

$$\underbrace{X(0)}_{0} + 1 + 2 + \dots + n = \frac{n \cdot (n+2)}{2} \in \Theta(n^2)$$

- $X(n) = X(n/3) + 1 \quad , \quad X(1) = 1$

$$n = 3^K$$

$$X(3^K) = X(3^{K-1}) + 1$$

$$= X(3^{K-2}) + 2$$

$$\vdots$$

$$= X(3^{K-i}) + i$$

$$\longrightarrow i \leftarrow K$$

$$X(1) + K = K + 1 = \log_3 n + 1 \in \Theta(\log n)$$

- $\sum_{i=2}^{n-1} \log_2 i^2$, find the order of growth

$$= 2 \cdot \sum_{i=2}^{n-1} \log_2 i = 2 \cdot \sum_{i=1}^{n} \log_2 i - 2 \cdot \log_2 n$$

$$2\Theta(n\log n) - 2\Theta(\log n) = \Theta(n \log n)$$